# ***Manos:*** A Benchmarking Client for ***RocksDB***

*(And Other Findings)*

# *Benchmarking Client*

Use our benchmarking client command line interface to...

- Create your custom workloads (e.g define proportion of range/point queries, deletes, inserts, updates, etc).
- Easily tune various RocksDB parameters .
- Run aggregated experiments.
- Run single experiments with using predefined experiment classes.
- Create your own experiments !

https://github.com/mdesilva/Manos

# How does RocksDB work ?

**In RocksDB, new writes are first sent to the *memtable* , an in-memory data structure, and optionally written to a *logfile*.**

As the data stored in the memtable is not persistent, writes can be sequentially appended to a *logfile* in the case a crash or system shutdown calls for a recovery.

**When the *memtable* is full, it is flushed to the *sstfile* (*static sorted file*), on storage, and the corresponding *logfile* can be deleted.**

# What are some knobs/parameters that we can tune ?

**Memtable size**

**Compaction strategy**

**Size of the LRU cache**

**Number of memtables**

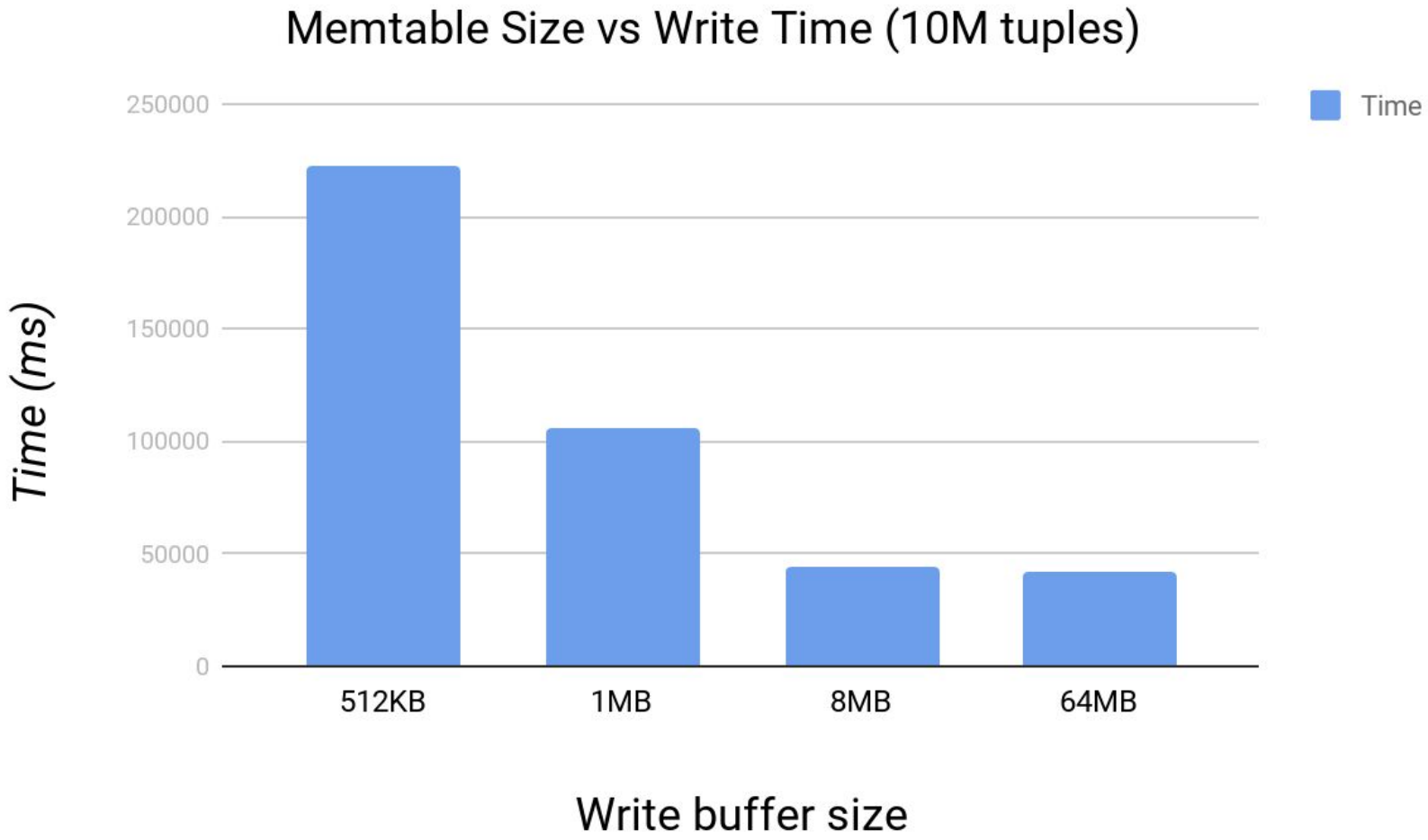
**Bloom filters**

**Allocation of Threads**

**Experiments performed on total dataset sizes from 50K tuples to 10M tuples*

# Memtable size

- Memtable: a set of in-memory write buffers
- Important to account for when dealing with bulk loading data
- **More** Memtable size -> **Less** flushing to disk -> **Increase** in write time

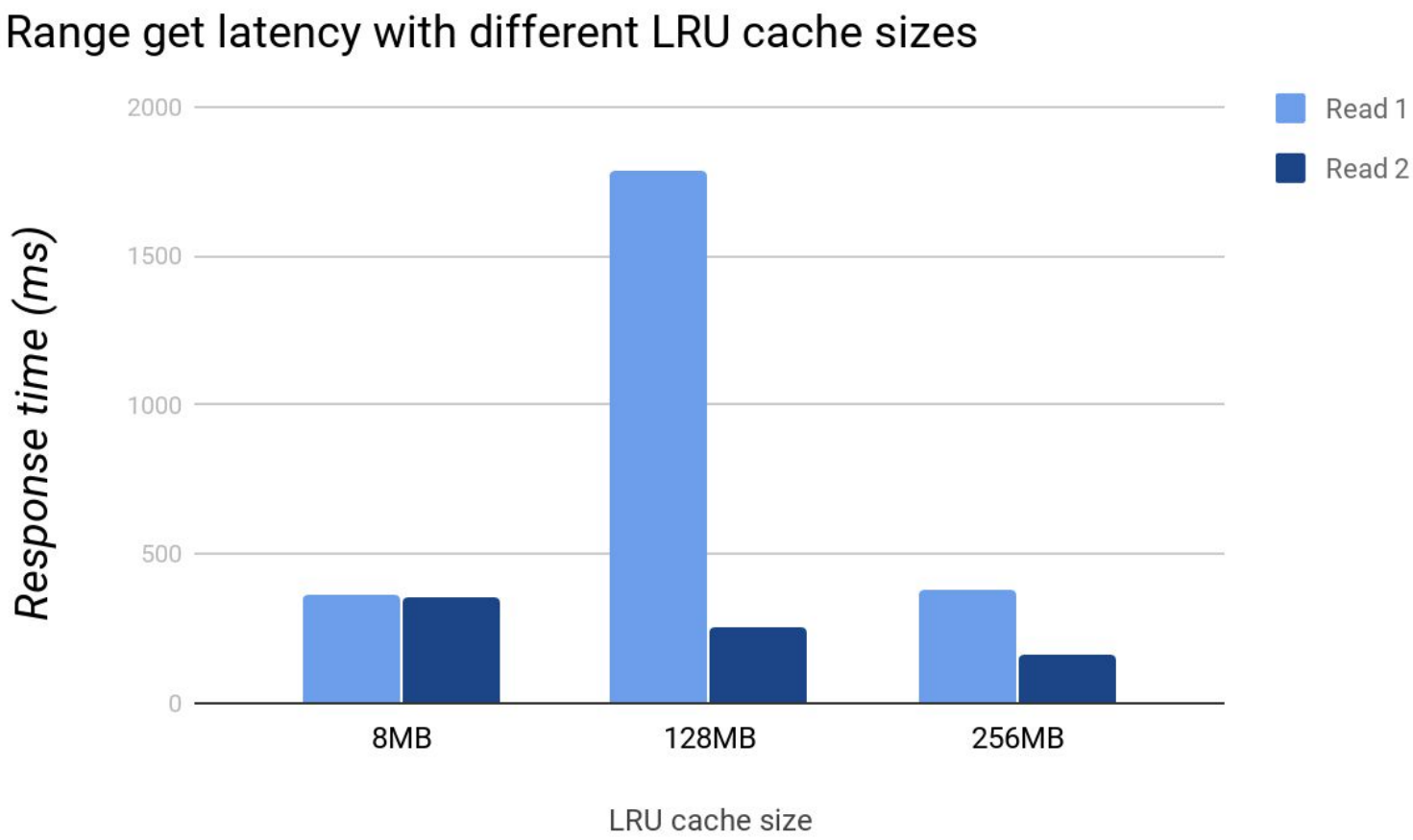

Memtable Size vs Write Time (10M tuples)
Time
250000
200000
150000
100000
50000
0
Time (ms)
512KB
1MB
8MB
64MB
Write buffer size

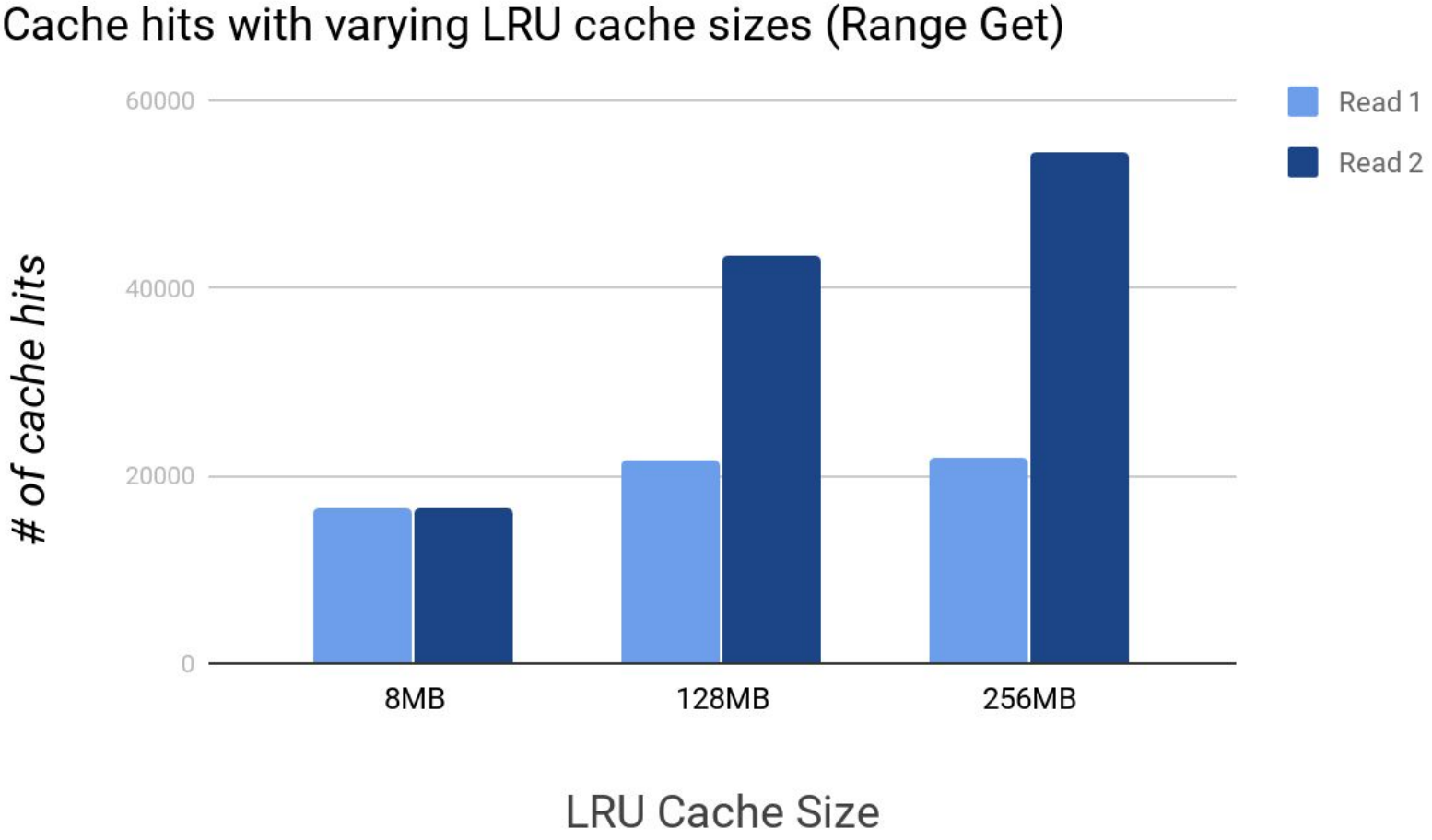
# LRU cache size

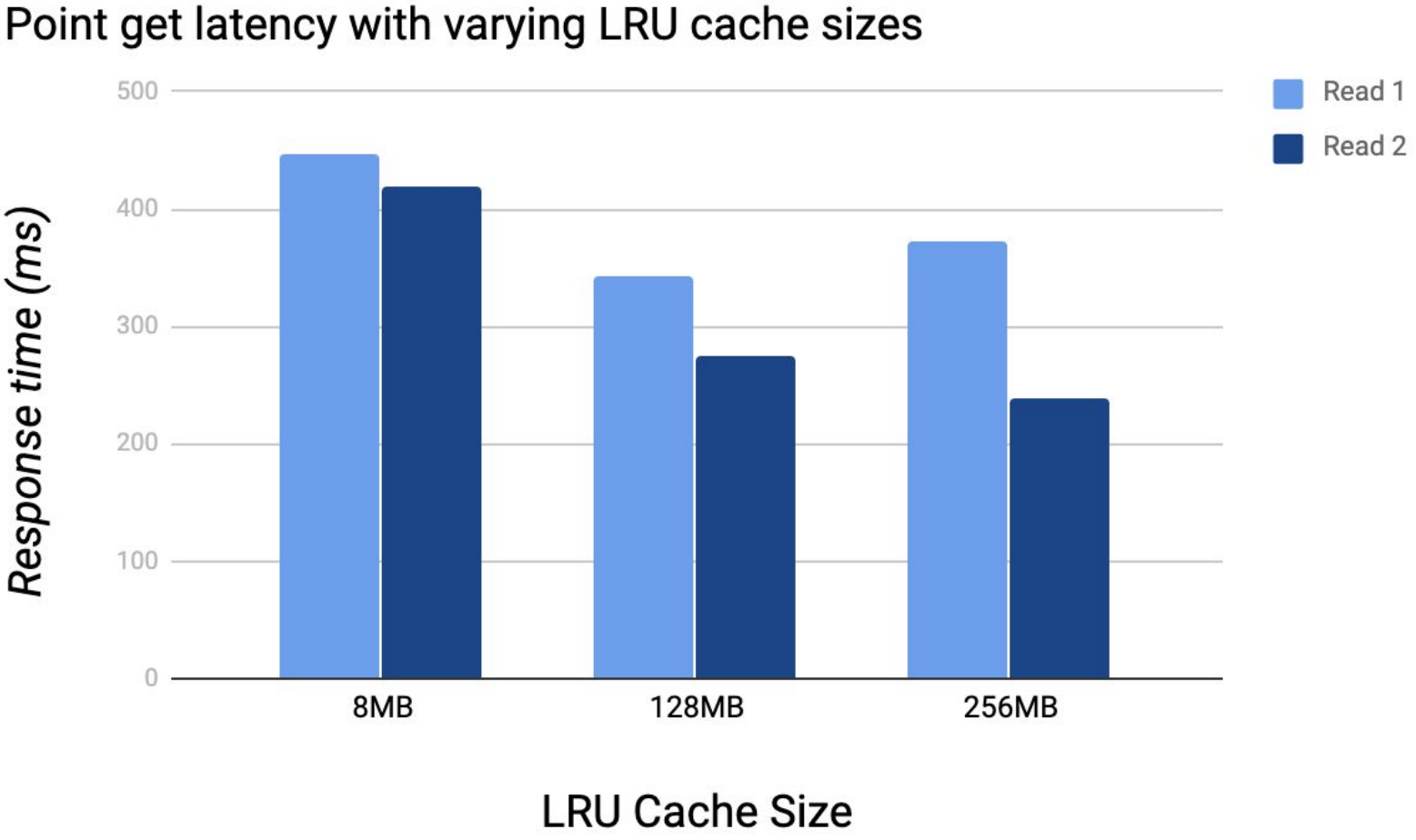
**RocksDB utilizes a block cache to cache data in-memory for reads that are served from disk.**

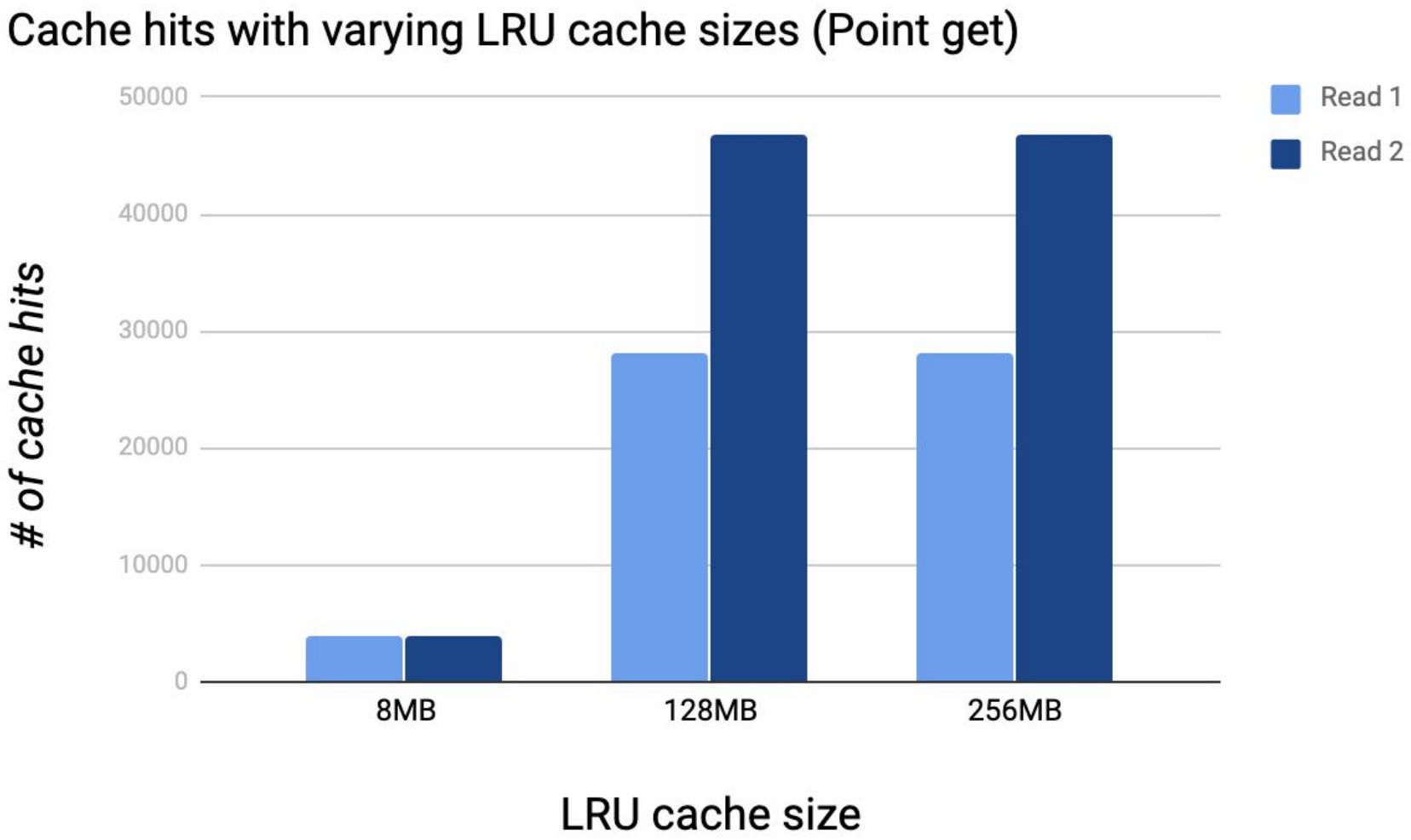
By default, it uses a LRU (least recently used) cache with a 8MB capacity.

**By increasing the size of the LRU cache, we can increase the number of cache hits, and decrease read times (read amplification).**


Range get latency with different LRU cache sizes
Response time (ms)
2000
1500
1000
500
0
8MB
128MB
256MB
LRU cache size
Read 1
Read 2


Cache hits with varying LRU cache sizes (Range Get)
60000
40000
20000
0
# of cache hits
Read 1
Read 2
8MB
128MB
256MB
LRU Cache Size


Point get latency with varying LRU cache sizes
500
400
300
200
100
0
Response time (ms)
Read 1
Read 2
8MB
128MB
256MB
LRU Cache Size


Cache hits with varying LRU cache sizes (Point get)
50000
40000
30000
20000
10000
0
# of cache hits
Read 1
Read 2
8MB
128MB
256MB
LRU cache size

# *Aggregated Experiments*

***Write Optimized*** parameters

LRU cache: Default

Memtable size: Default

Bloom filter size: Default

# of memtables: 5

Universal style compaction

***Read Optimized*** *parameters*

LRU cache: 512MB

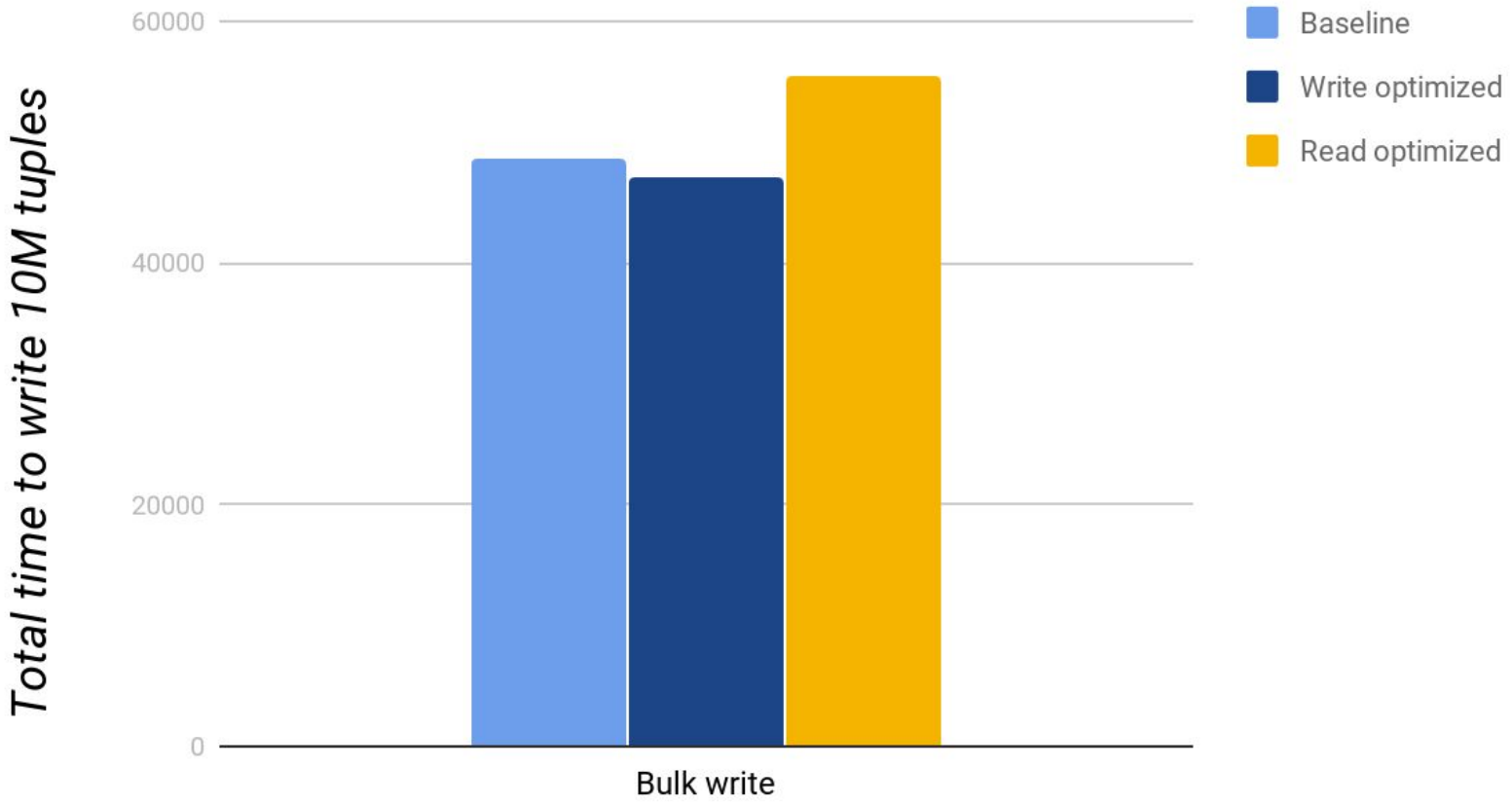
Memtable size: 256MB

Bloom filter size: 100 bits

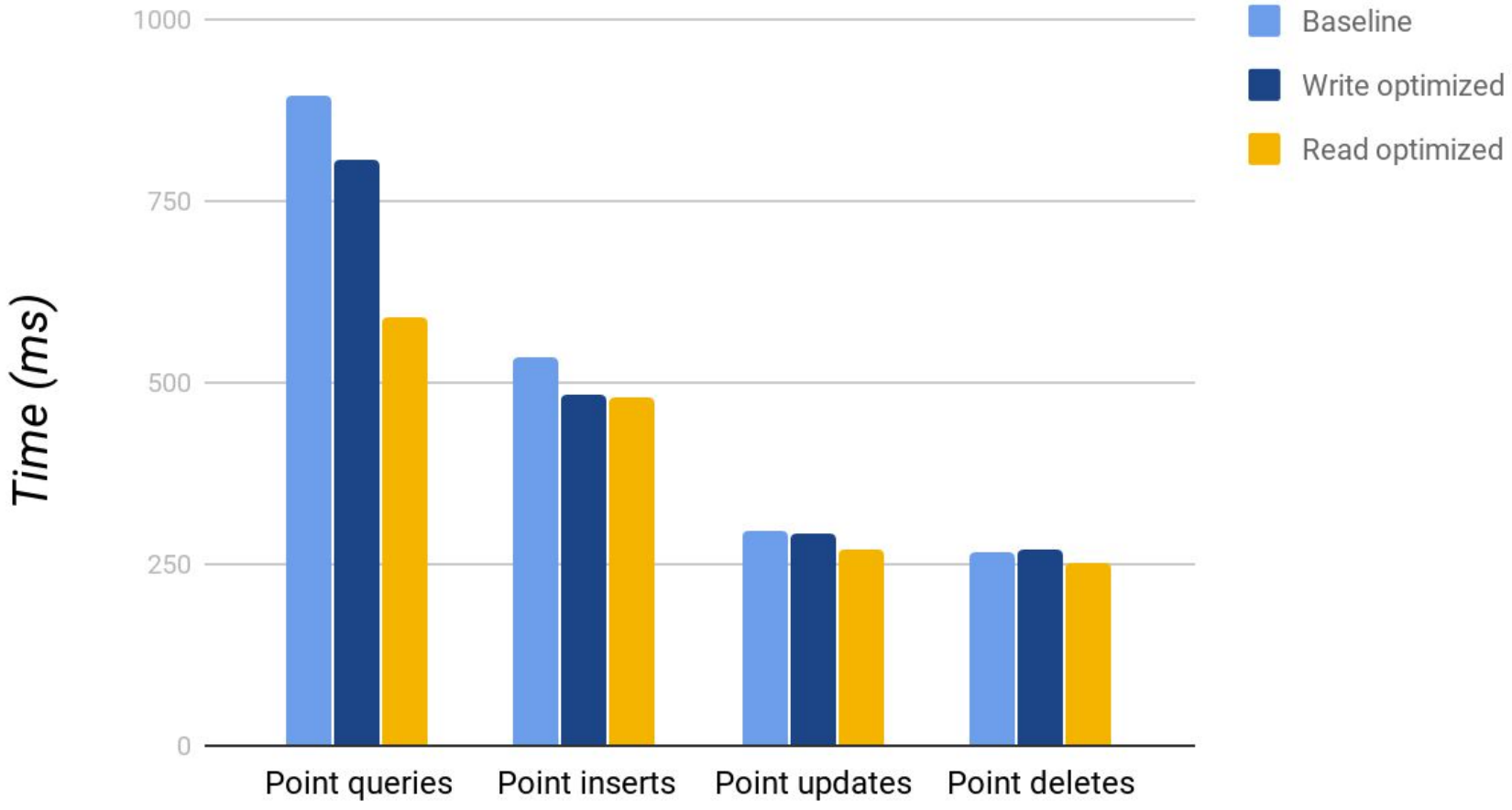
# of memtables: Default

Level style compaction

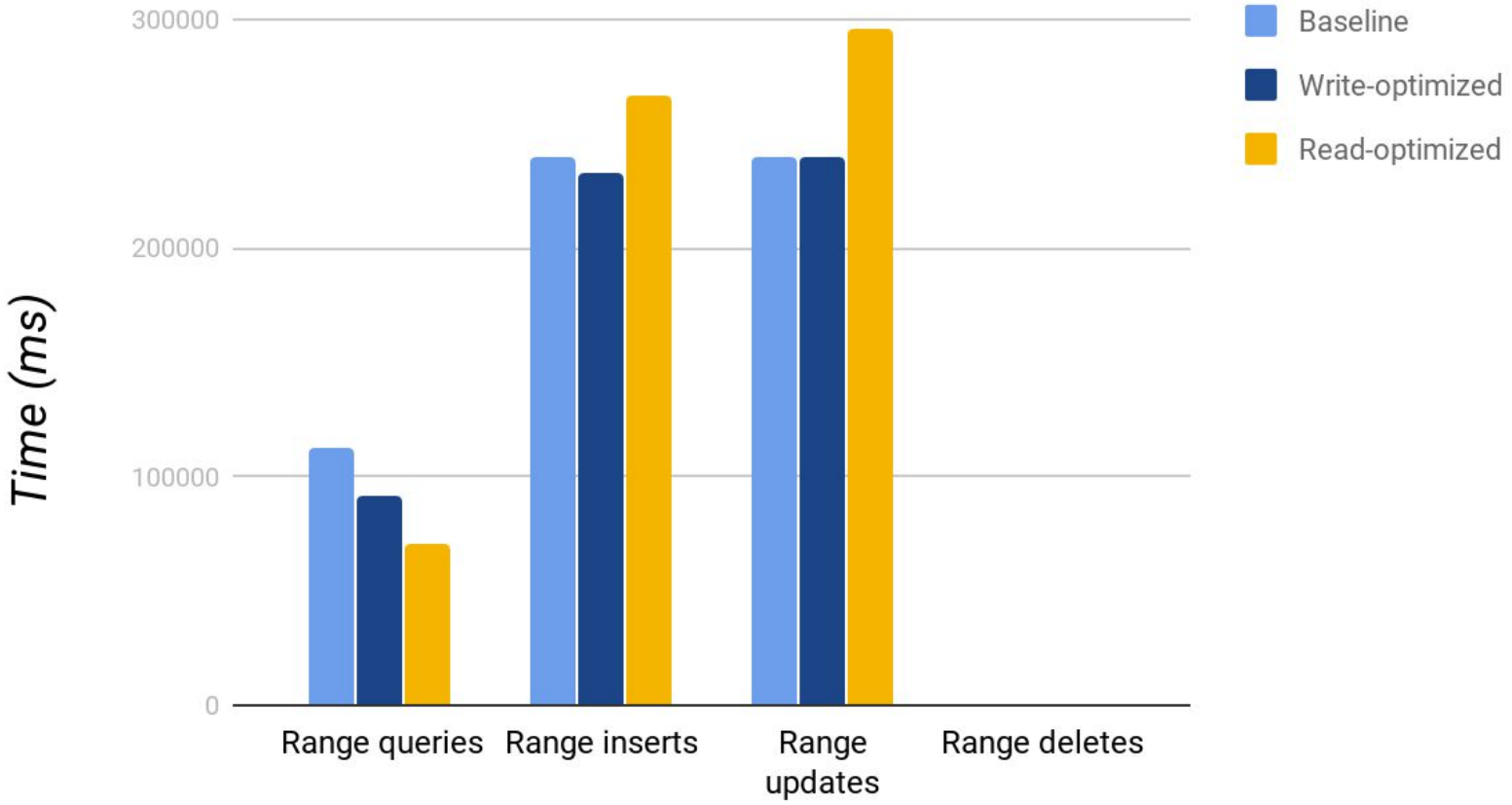
# Bulk Write Time



# Point operations



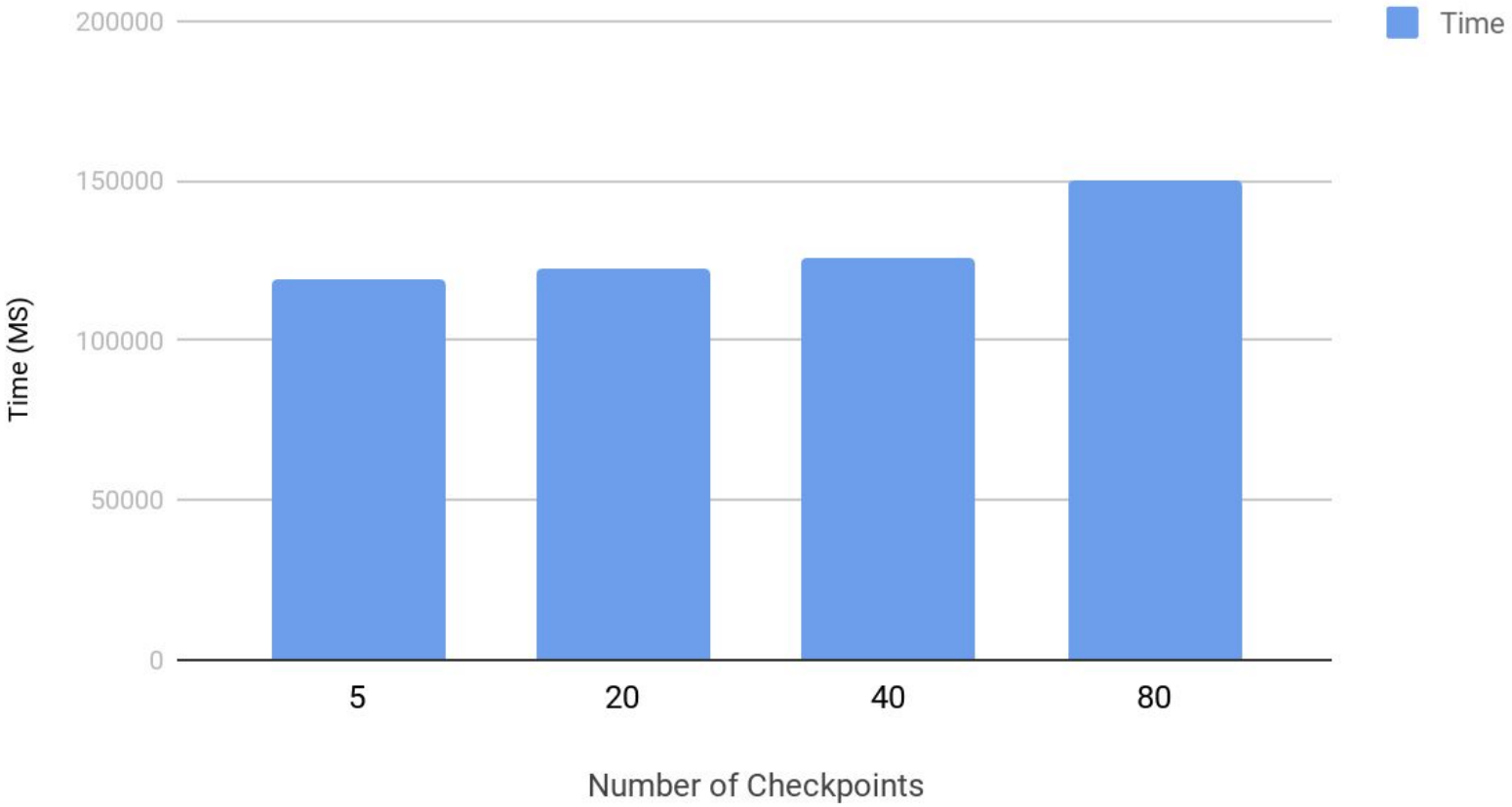
# Range operations



# RocksDB Supporting Apache Flink

- Flink is an open source Apache project designed for handling data streams.
- In production settings, it relies on RocksDB to support it's data storage needs
- Latency vs Accuracy when determining how many checkpoints to implement?

# Number of Checkpoints vs Write Time (10M Tuples)

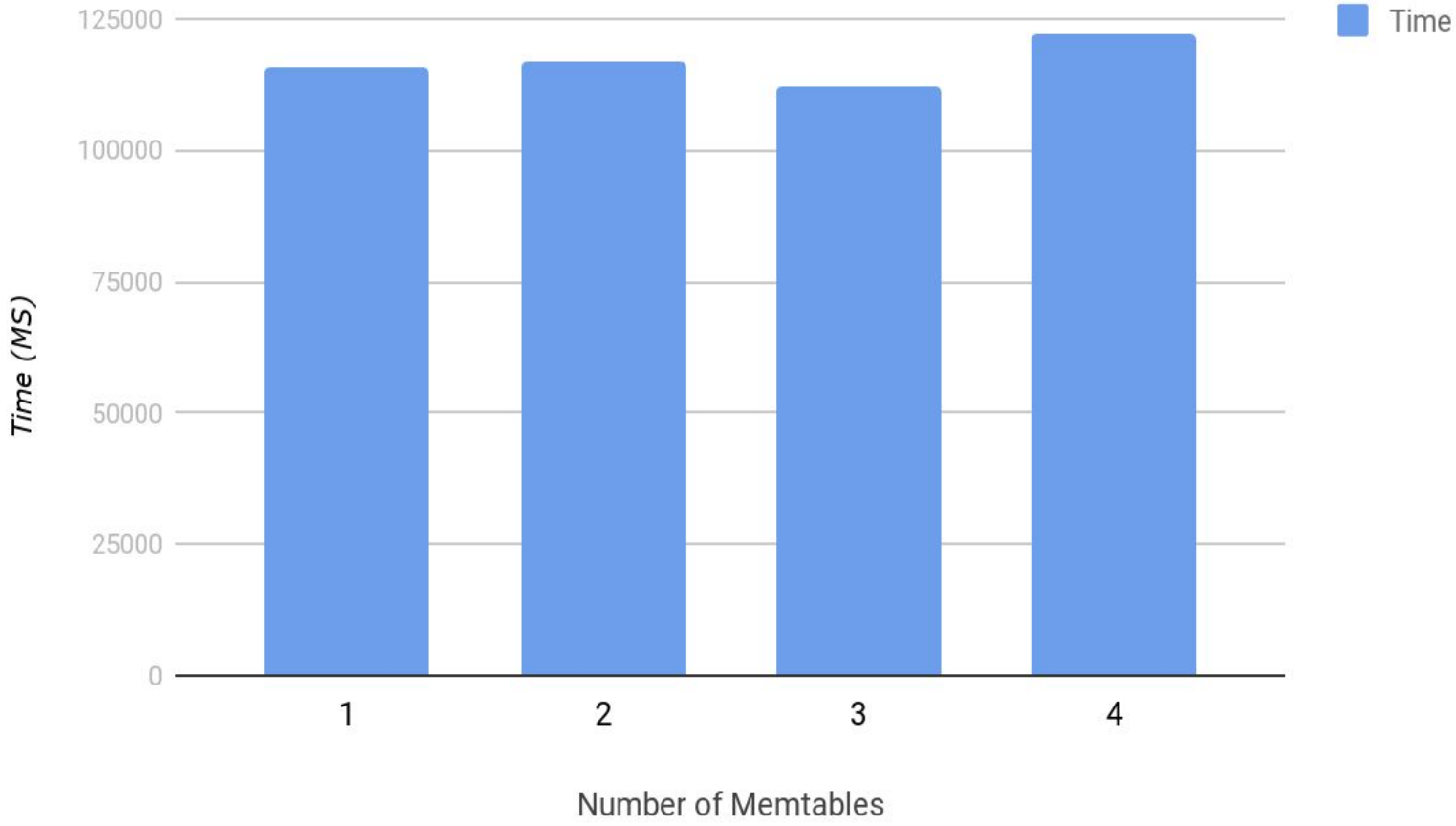


# Additional Slides

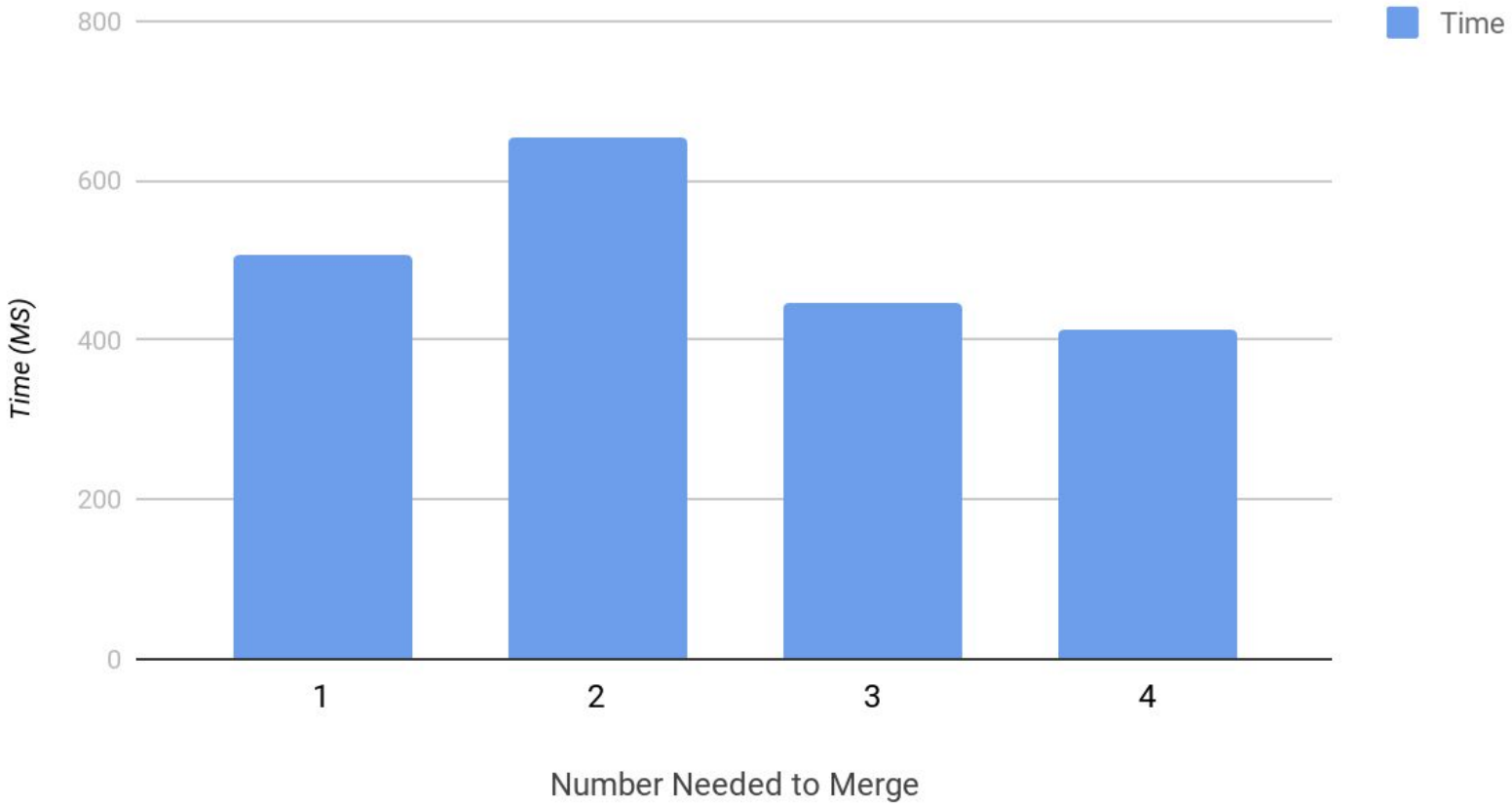
# Number of Memtables

All writes to RocksDB start by being inserted into a memtable. Once the current memtable is full, a new one is created, and the former becomes read only (immutable). The immutable memtables are then queued to be flushed to storage. ***Writes can be stalled if the current memtable is filled and there's no allocation for more, while read performance can deteriorate if the required amount to merge is set too low.***

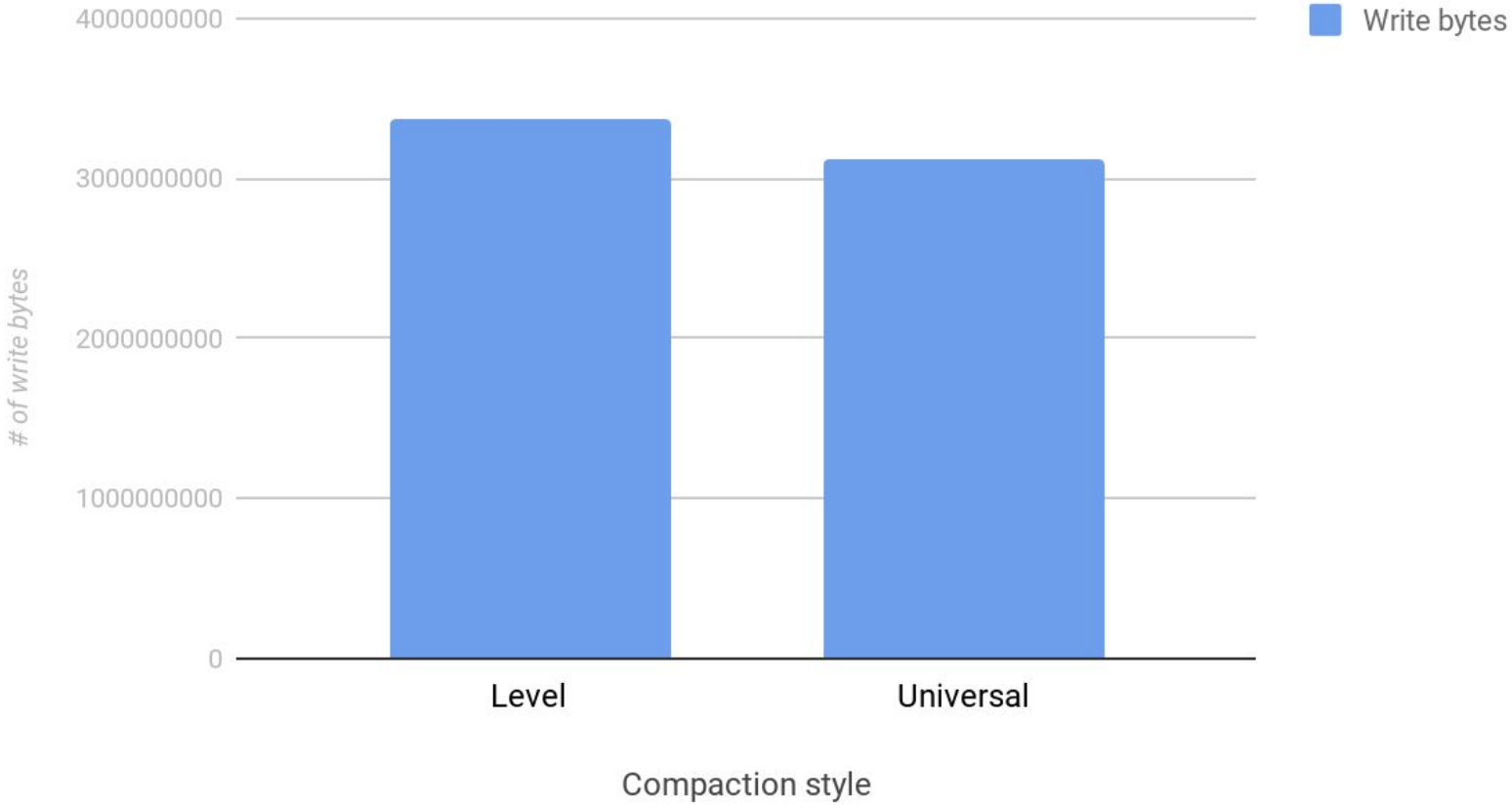
# Number of Memtables vs Write Time (10M Tuples)



# Min Number of Memtables to Merge vs Read Time

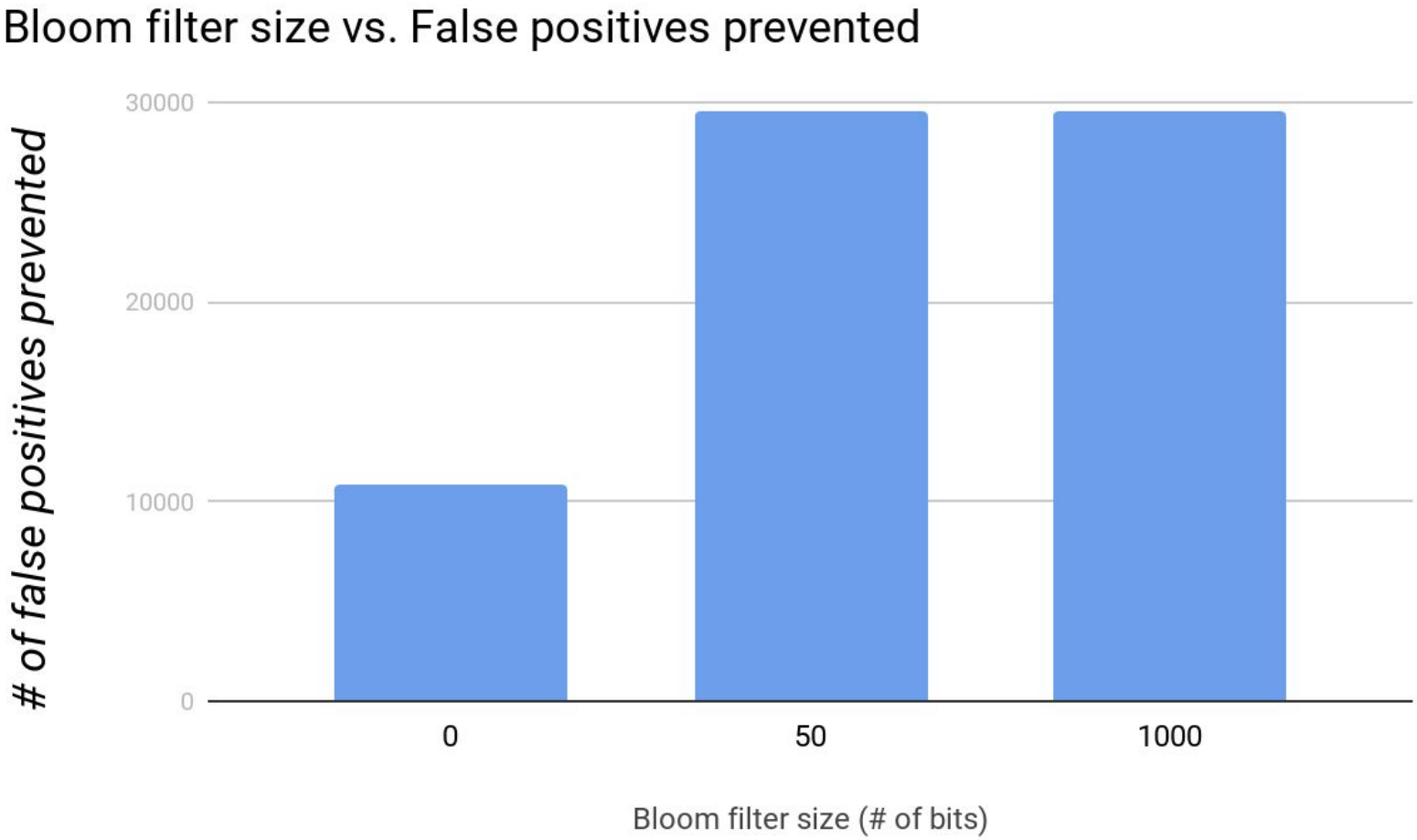


# Write bytes vs Compaction style



# Bloom filters

- Keys are stored across SST files once merged to disk
- Reads may have to traverse multiple files to get a value.
- Bloom filters are bit arrays that can determine if a key **exists** in a SST file.
- **Bloom filters can prevent extra reads of files**
- **Especially useful for point lookups.**

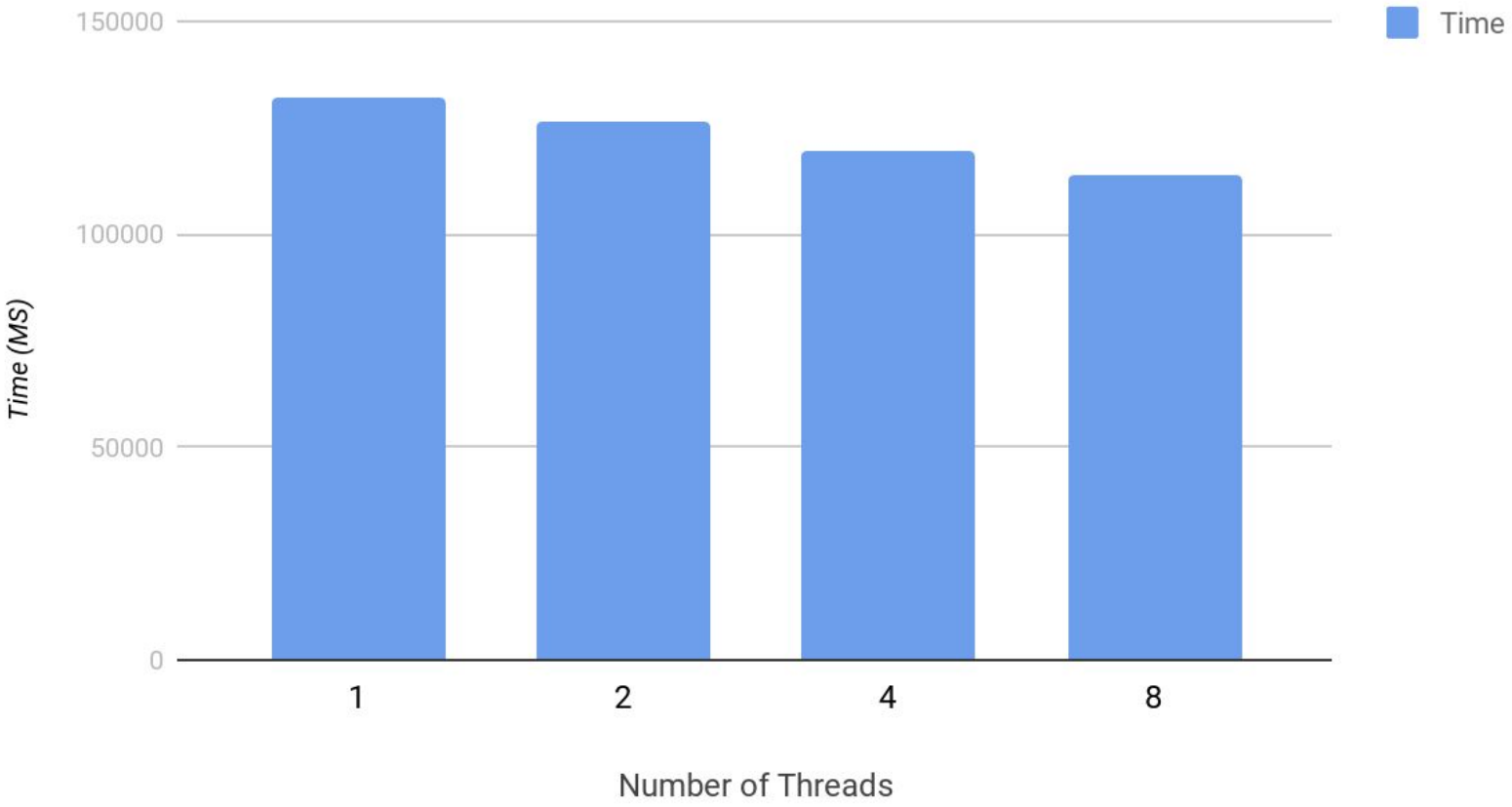



**We found that increasing the bloom filter size did not significantly reduce read times, even though the number of false positives prevented increased.**

# Allocation of Threads

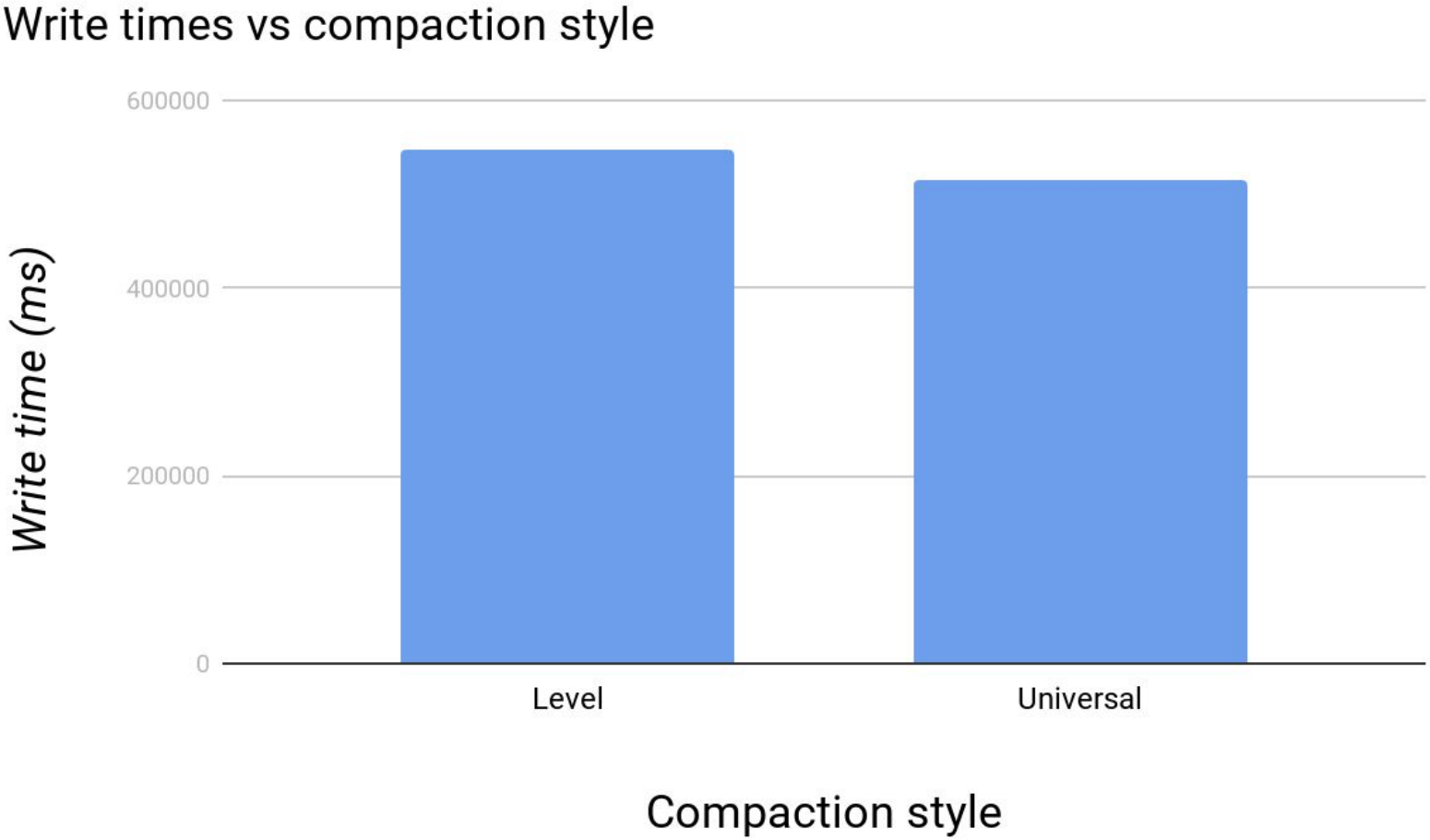
- There are two main processes in RocksDB
- *Flush*
- *Compaction*
- **More compaction threads, slower writes**

# Number of Threads vs Time



# Compaction style

- RocksDB supports two types of compaction.
- **Level-style** takes up less space - optimizing read amplification
- **Universal style** take more temporary space - optimzing write amplification


Write times vs compaction style
600000
400000
200000
0
Write time (ms)
Level
Universal
Compaction style

# What is amplification ?

Write amplification - # of bytes written to storage: # of bytes written to database

Read amplification - # of disk reads: query

Space amplification - size of database files on disk: data size